SILICON VALLEY
DATA SCIENCE

SPARK
SUMMIT
2017

Data Wrangling with PySpark for Data Scientists Who Know Pandas

Dr. Andrew Ray

ANDREW RAY

@collegeisfun

Spark contributor

- SparkSQL
- GraphX
- PySpark

Principal Data Engineer @ SVDS

Previously Data Sci. @ Walmart

PhD Mathematics @ Nebraska

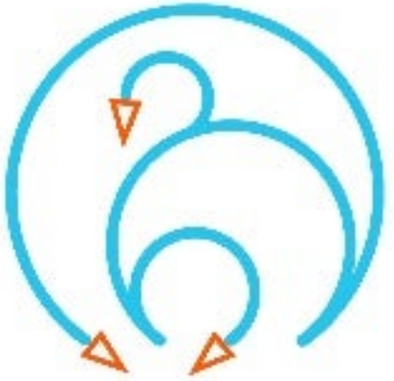
Silicon Valley Data Science is a boutique consulting firm focused on transforming your business through data science and engineering.

OUR SERVICES

DATA STRATEGY

ARCHITECTURE

AGILE ENGINEERING



AGILE DATA SCIENCE

COME SEE US & SAY HELLO!

Wednesday, June 7

- **Write Graph Algorithms Like a Boss** with Andrew Ray (3:20 pm)

To view SVDS speakers and scheduling,
or to receive a copy of our slides, go to:

www.svds.com/SparkSummit17

OUTLINE

- Why PySpark?
- Primer
- Setup & Run
- vs. Pandas
- Best Practices


PySpark
WHY?
@SVDataScience

WHY?

Do you:

1. Already know Python & Pandas?
2. Love DataFrames?
3. Want to work with Big Data?

Then PySpark is the answer!

WHAT DO I GET WITH PYSPARK?

Gain

- Work with big data
- Native SQL
- Decent documentation

Lose

- Amazing documentation
- Easy plotting
- Indices

PYSPARK

A (quick) Primer

Apache Spark is a fast and general engine
for large-scale data processing.

PRIMER

Distributed compute

- YARN, Mesos, Standalone cluster

Abstractions

- RDD—distributed collection of objects
- Dataframe—distributed dataset of tabular data.
  - Integrated SQL
  - ML Algorithms

IMPORTANT CONCEPTS

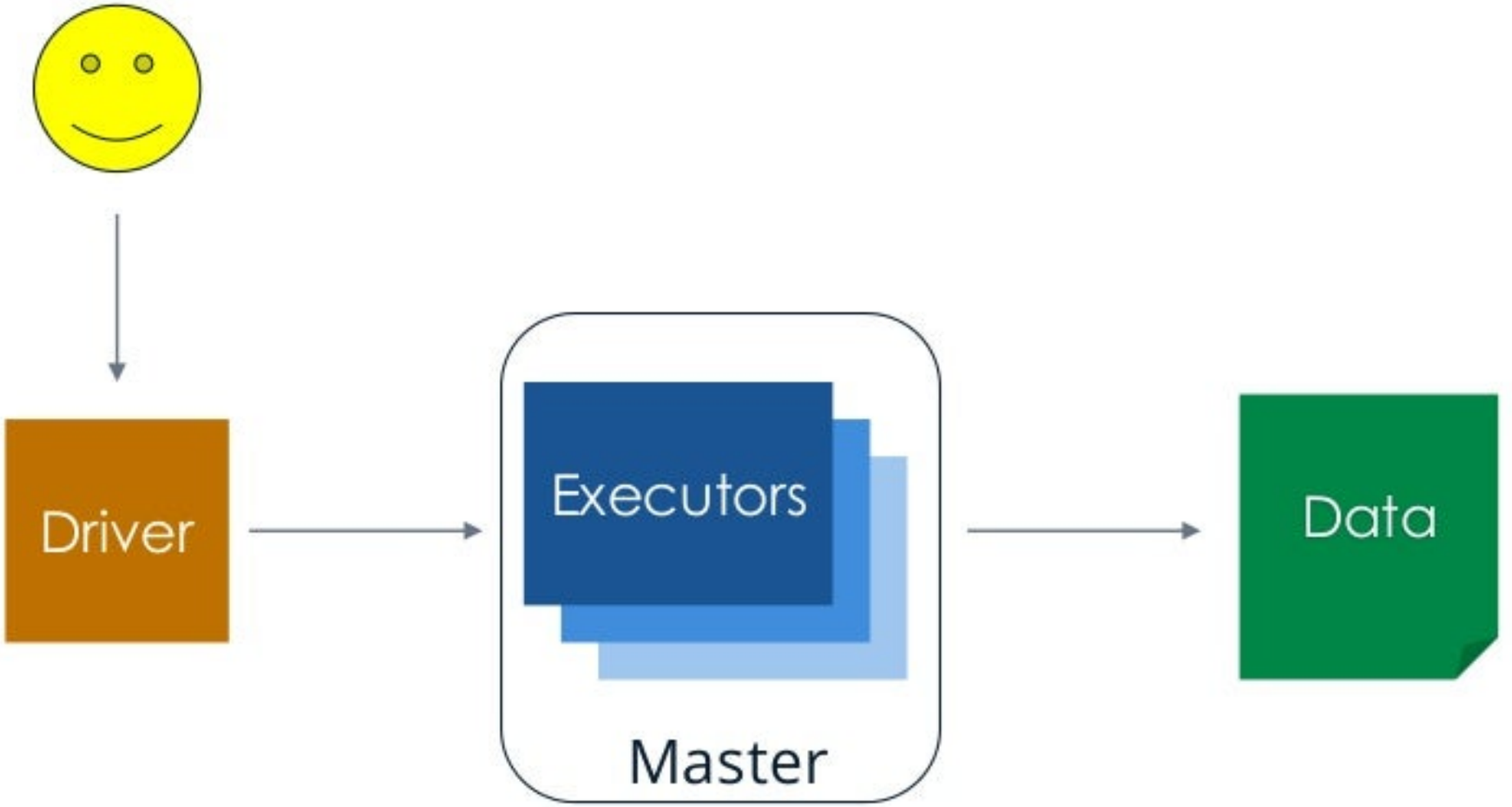
Immutable

- Changes create new object references
- Old versions are unchanged

Lazy

- Compute does not happen until output is requested

ARCHITECTURE




Setup and Run
PYSPARK
@SVDataScience

SETUP

- Option 1:
  - https://spark.apache.org/downloads.html
  - `tar -xzf spark-2.1.1-bin-hadoop2.7.tgz`
  - `PATH="$PATH:$(pwd)/spark-2.1.1-bin-hadoop2.7/bin"`
- Option 2:
  - `conda install -c conda-forge pyspark=2.1.1`
- (Future) Option 3:
  - `pip install pyspark`

RUN

A. `pyspark`

B. `PYSPARK_DRIVER_PYTHON=ipython pyspark`

C.
```
PYSPARK_DRIVER_PYTHON=jupyter \
PYSPARK_DRIVER_PYTHON_OPTS=notebook \
pyspark
```



PYSPARK
vs. Pandas
@SVDataScience

LOAD CSV

Pandas

```
df = pd.read_csv("mtcars.csv")
```

PySpark

~~df = spark.read.csv("mtcars.csv")~~

LOAD CSV

Pandas

```
df = pd.read_csv("mtcars.csv")
```

PySpark

```
df = spark.read \
  .options(header=True, inferSchema=True) \
  .csv("mtcars.csv")
```


VIEW DATAFRAME

Pandas

```
df
df.head(10)
```

PySpark

```
df.show()
df.show(10)
```


COLUMNS AND DATA TYPES

Pandas

```
df.columns
df.dtypes
```

PySpark

```
df.columns
df.dtypes
```


RENAME COLUMNS

Pandas

```
df.columns = ['a', 'b', 'c']
df.rename(columns = {'old': 'new'})
```

PySpark

```
df.toDF('a', 'b', 'c')
df.withColumnRenamed('old','new')
```


DROP COLUMN

Pandas

```
df.drop('mpg', axis=1)
```

PySpark

```
df.drop('mpg')
```


FILTERING

Pandas

```
df[df.mpg < 20]
df[(df.mpg < 20) & (df.cyl == 6)]
```

PySpark

```
df[df.mpg < 20]
df[(df.mpg < 20) & (df.cyl == 6)]
```


ADD COLUMN

Pandas

```
df['gpm'] = 1 / df.mpg
```

PySpark

```
df.withColumn('gpm', 1 / df.mpg)
```


FILL NULLS

Pandas

```
df.fillna(0)
```
← Many more options

PySpark

```
df.fillna(0)
```


AGGREGATION

Pandas

```
df.groupby(['cyl', 'gear']) \
  .agg({'mpg': 'mean', 'disp': 'min'})
```

PySpark

```
df.groupby(['cyl', 'gear']) \
  .agg({'mpg': 'mean', 'disp': 'min'})
```


OK we get the point

STANDARD TRANSFORMATIONS

Pandas

```
import numpy as np
df['logdisp'] = np.log(df.disp)
```

PySpark

```
import pyspark.sql.functions as F
df.withColumn('logdisp', F.log(df.disp))
```


KEEP IT IN THE JVM

```
import pyspark.sql.functions as F
```

```
AutoBatchedSerializer collect_set expr length rank substring Column column ctorial
levenshtein regexp_extract substring_index Dataame concat rst lit regexp_replace sum
PickleSerializer concat_ws oor locate repeat sumDistinct SparkContext conv rmat_number log
reverse sys StringType corr rmat_string log10 rint tan UserDenednction cos om_json log1p
round tanh abs cosh om_unixtime log2 row_number toDegrees acos count om_utc_timestamp
lower rpad toRadians add_months countDistinct get_json_object lpad rtrim to_date
approxCountDistinct covar_pop greatest ltrim second to_json approx_count_distinct
covar_samp grouping map sha1 to_utc_timestamp array crc32 grouping_id math sha2 translate
array_contains create_map hash max shiLe trim asc cume_dist hex md5 shiRight trunc ascii
current_date hour mean shiRightUnsigned udasin current_timestamp hypot min signum unbase64
atan date_add ignore_unicode_prex minute sin unhex atan2 date_rmat initcap
monotonically_increasing_id since unix_timestamp avg date_sub input_le_name month sinh
upper base64 datedi instr months_between size v bin dayoonth isnan nanvl skewness var_pop
bitwiseNOT dayoear isnull next_day sort_array var_samp blacklist decode json_tuple ntile
soundex variance broadcast degrees k percent_rank spark_partition_id weekoear bround
dense_rank kurtosis posexplode split when cbrt desc lag pow sqrt window ceil encode last
quarter stddev year coalesce exp last_day radians stddev_pop col explode lead rand
stddev_samp collect_list expm1 least randn struct
```


ROW CONDITIONAL STATEMENTS

Pandas

```
df['cond']=df.apply(lambda r:
  1 if r.mpg > 20 else 2 if r.cyl == 6 else 3,
  axis=1)
```

PySpark

```
import pyspark.sql.functions as F
df.withColumn('cond', \
  F.when(df.mpg > 20, 1) \
   .when(df.cyl == 6, 2) \
   .otherwise(3))
```


PYTHON WHEN REQUIRED

Pandas

```
df['disp1'] = df.disp.apply(lambda x: x+1)
```

PySpark

```
import pyspark.sql.functions as F
from pyspark.sql.types import DoubleType
fn = F.udf(lambda x: x+1, DoubleType())
df.withColumn('disp1', fn(df.disp))
```


MERGE/JOIN DATAFRAMES

Pandas

```
left.merge(right, on='key')
left.merge(right, left_on='a', right_on='b')
```

PySpark

```
left.join(right, on='key')
left.join(right, left.a == right.b)
```


PIVOT TABLE

Pandas

```
pd.pivot_table(df, values='D', \
  index=['A', 'B'], columns=['C'], \
  aggfunc=np.sum)
```

PySpark

```
df.groupBy("A", "B").pivot("C").sum("D")
```


SUMMARY STATISTICS

Pandas

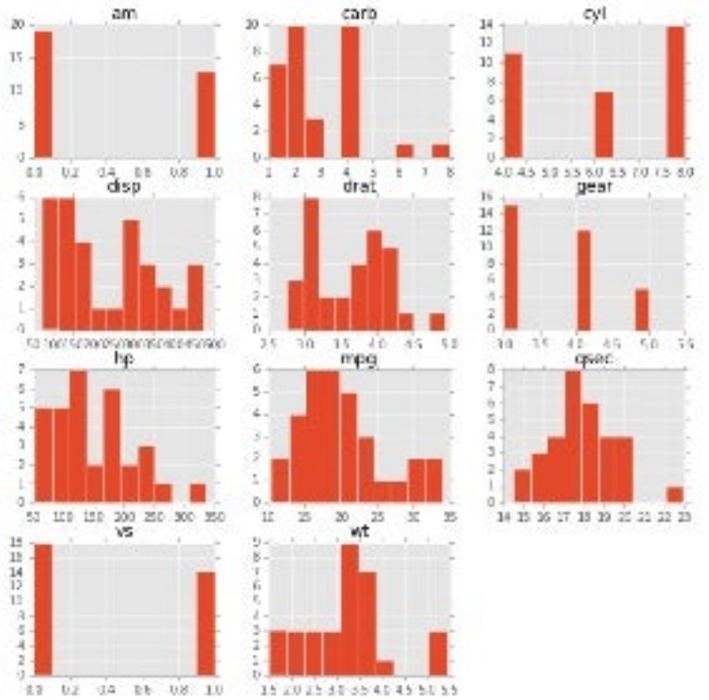
```
df.describe()
```

PySpark

```
df.describe().show()
```
(only count, mean, stddev, min, max)

```
df.selectExpr(
  "percentile_approx(mpg, array(.25, .5, .75)) as mpg"
  ).show()
```


HISTOGRAM



Pandas

```
df.hist()
```

PySpark

```
df.sample(False, 0.1).toPandas().hist()
```


SQL

Pandas

n/a

PySpark

```
df.createOrReplaceTempView('foo')
df2 = spark.sql('select * from foo')
```



PYSPARK
Best Practices
@SVDataScience

MAKE SURE TO

- Use `pyspark.sql.functions` and other built in functions.
- Use the same version of python and packages on cluster as driver.
- Check out the UI at http://localhost:4040/
- Learn about SSH port forwarding
- Check out Spark MLlib
- RTFM: https://spark.apache.org/docs/latest/

THINGS NOT TO DO

- Try to iterate through rows
- Hard code a master in your driver
  - Use `spark-submit` for that
- `df.toPandas().head()`
  - instead do: `df.limit(5).toPandas()`

IF THINGS GO WRONG

- Don’t panic!
- Read the error
- Google it
- Search/Ask Stack Overflow (tag apache-spark)
- Search/Ask the user list: user@spark.apache.org
- Find a bug? Make a JIRA ticket: https://issues.apache.org/jira/browse/SPARK/

SILICON VALLEY
DATA SCIENCE

info@svds.com
@SVDataScience

THANK YOU

Andrew Ray
andrew@svds.com
@collegeisfun